AD-A264 993

ENTATION PAGE

Form Approved
OMB No. 0704-0188

PORT DATE	3. REPORT TYPE AND DATES COVERED
February 1993	professional paper

4. TITLE AND SUBTITLE

EFFECTIVE SOFTWARE REUSE IN AN EMBEDDED REALTIME SYSTEM

5. FUNDING NUMBERS

PR: CM16
PE: OMN
WU: DN213075

6. AUTHOR(S)

B. Barlin and J. M. Lawler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division
San Diego, CA 92152-5001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Space and Naval Warfare Systems Command
Washington, DC 20363-5100

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Authorized for public release; distribution is unlimited.

12b. DISTRIBUTION

13. ABSTRACT *(Maximum 200 words)*

The Submarine Message Buffer (SMB) is a real time embedded message processing system developed at the Naval Command, Control and Ocean Surveillance Center, Research, Development, Test and Evaluation Division (NRaD). The SMB is sponsored by the Space and Naval Warfare Systems Command (SPAWAR) to support modernization of SSN (Los Angeles and Seawolf class submarines) radio rooms. The development strategy adopted for the SMB concentrated on the reuse of Ada software. This paper will focus on the design, processes and methodologies, which were used in the development of this system. Metrics will also be provided showing why this system has been identified as an Ada success story by the Ada Joint Program Office among others.

~~Published in *Proceedings TRI-Ada '92*, November 1992.~~

14. SUBJECT TERMS

communications
Verdin
VLF communications
ISABPS
NISBS
SMB

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

Effective Software Reuse in an Embedded Real-Time System

B. Barlin
Naval Command, Control and Ocean Surveillance Center
Research, Development, Test and Evaluation Division
Open Systems Development Branch
271 Catalina Blvd
San Diego Ca 92152-5000

J.M. Lawler
MITRE Corporation
Naval Warfare Systems
271 Catalina Blvd
San Diego Ca 92152-6550

Abstract

The Submarine Message Buffer (SMB) is a real time embedded message processing system developed at the Naval Command, Control and Ocean Surveillance Center, Research, Development, Test and Evaluation Division (NRaD). The SMB is sponsored by the Space and Naval Warfare Systems Command (SPAWAR) to support modernization of SSN (Los Angels and Seawolf class submarines) radio rooms. The development strategy adopted for the SMB concentrated on the reuse of Ada software. This paper will focus on the design, processes and methodologies, which were used in the development of this system. Metrics will also be provided showing why this system has been identified as an Ada success story by the Ada Joint Program Office among others [1,2].

Introduction

Figure 1 is a block diagram of the SMB. The SMB is a real time message processor that can receive and transmit messages from various RF modems on the SSN via four AN/UGC-136C(X) Teletypes while simultaneously performing other standard message processing functions. These functions include identifying the message type, parsing out key message elements and inputting the messages into a database. The SMB also enables an operator to perform standard message management activities according to standard Naval procedures. These activities include maintenance of message audit trails, providing an automated radio log, word processing and message template forms for message composition (ACP-126, ACP-127, JANAP-128, JINTACCS). The SMB can also get messages through the operator interface or floppy disk.

The SMB is targeted for an 80386 Personal Computer with I/O boards which reside on the ISA bus. These I/O boards are single board computers based on Intel 80186 processors which provide four full duplex channels per board. They are programmable and allow for the use of various communication interfaces and protocols.

Microsoft's MS-DOS™ was chosen as the operating system to maximize the use of commercial-off-the-shelf software and reuse of government owned software.

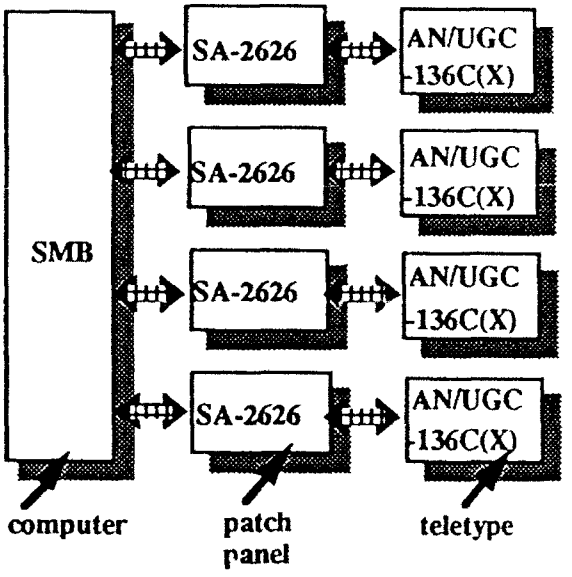


Figure 1. SMB Block Diagram

Many of the functions required of the SMB are similar to other military message processing systems already developed and fielded. Further, one of the mission focus areas for the NRaD Operational Systems Branch is the development and maintenance of message processing systems. For these reasons a search to identify a library of reusable message processing software was initiated.

The initial search identified a number of candidates whose specifications were put into the library. These candidates were evaluated by a preset strategy [3]. This evaluation resulted in identifying two candidate Ada programs, the JINTACCS Automated Message Editing System (JAMES) and the NATO Interoperable Submarine Broadcast System (NISBS) that were to be reused for the

SMB. Together, these two programs provided 80% of the desired functionality of the SMB.

Design

The SMB is composed of the following three subsystems: User Interface, Message Processing, and Input/Output Subsystems. JAMES was the source of the User Interface and Message Processing Subsystems software. The Input/Output Subsystem software, called IO_TOOLS, was obtained from NISBS. JAMES is similar to character based commercial word processing programs for MS-DOS. It provides the capability to edit, retrieve, create, print and save messages in files. Also, like commercial word processors for MS-DOS, it has no real time, multiprocessing capabilities. The IO_TOOLS provide the real time capability and are the key to the reuse success on this project.

The IO_TOOLS were developed for the NISBS program under an Ada Technology Insertion Program (ATIP) grant from the Ada Joint Program Office (AJPO). These tools were designed for maximum reusability and leverage PC technology to provide an inexpensive yet very powerful real time multichannel system. The overall design of the IO_TOOLS was based on a layered approach to system composition as shown in Figure 2. Each layer encapsulates a very well defined set of functionality that isolates the next higher level from details not needed at that level. This enhances the reusability of the code at all levels.

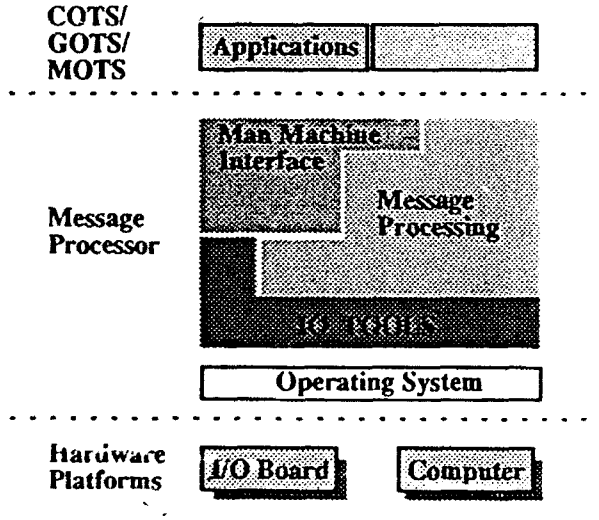


Figure 2. System Layering

This layering approach allows Commercial-Off-The-Shelf (COTS), Government-Off-The-Shelf (GOTS) and Military-Off-The-Shelf (MOTS) Ada applications, such as word processors, to be integrated into the system, without the need for modification of the application itself. This is accomplished through the use of Ada Tasking as shown in Figure 3. The IO_TOOLS and the application are run as separate tasks. These tasks have no need to communicate directly nor rendezvous. The IO_TOOLS insures that incoming message traffic is assembled into files and are inserted into the appropriate directory in format which the application can manipulate or import. From the viewpoint of the application it is as if it is running on the system by itself, using its normal methods of non-real-time I/O.

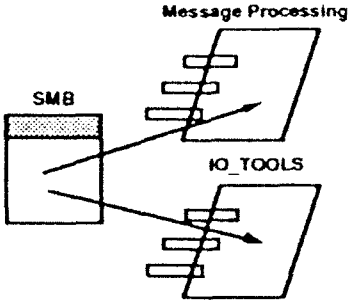


Figure 3. Booch Diagram of SMB Tasking

Object Oriented Design was used to provide flexibility in programming the embedded I/O boards for connecting different types of I/O devices to the SMB [4]. The details of the embedded processor's operation was encapsulated into two objects, the SCC_PKG and the DMA_PKG. Figure 4 is a Booch diagram of the IO_TOOLS which includes these two objects. Figure 5 shows the IO_TOOLS physical allocation of the software between the host and embedded processors within the hardware.

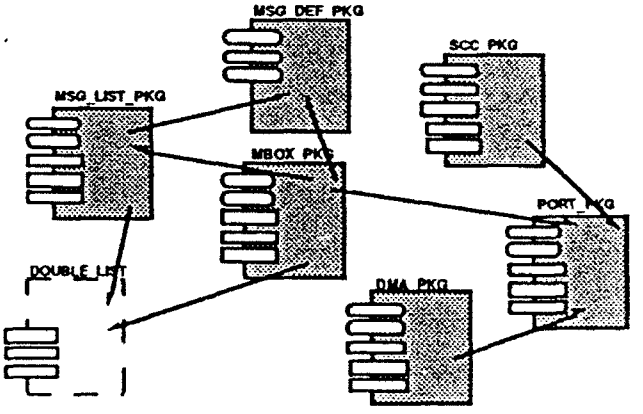


Figure 4. Booch Diagram of the IO_TOOLS

The IO_TOOLS has a message passing scheme to transfer commands and data between the host and the I/O boards. The board to host interface is performed through shared memory. The primary packages of the IO_TOOLS are described as follows.

MSG_DEF_PKG: Contains type definitions for messages that will be passed between the IO_TOOLS software on the I/O board and the host software on the PC. Additional IO_TOOLS capabilities can be provided by expanding the message definitions and providing the appropriate processing.

MBOX_PKG: Defines the types and operations to implement mailboxes for the IO_TOOLS and the host software. Messages are delivered and removed from a specific "users" mailbox. Users are the IO_TOOLS and the host software. When a message is read from a mailbox, the receiver acts on the message. Commands are acted on and status information is processed accordingly.

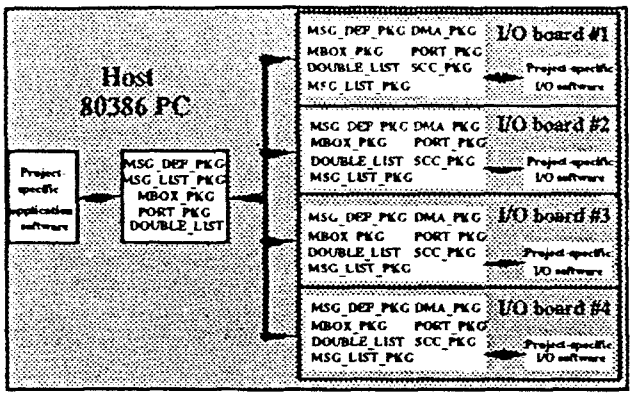


Figure 5. Software to Hardware Allocation

SCC_PKG: Provides an interface to the Zilog Z8530 Serial Communications Controller (SCC) chip on the I/O boards. It further provides a high level interface to the low level chip operations. Every primary operation of the chip is represented even though only a subset is used.

DMA_PKG: Provides an interface to the Intel 8237A Direct Memory Access (DMA) chip on the I/O board. It also provides a high level interface to the low level chip operations. Every primary operation of the chip is represented even though only a subset is used.

PORT_PKG: Contains the operations to read and write data to and from the specified port on a specified I/O board. Overloaded operations are provided to read and write data of different sizes.

DOUBLE_LIST: Provides operations to manipulate the abstract data type LIST_ID, which is a double linked list of objects. DOUBLE_LIST is a generic package and the type of the list object is declared at instantiation. It provides operations to add objects to, delete objects from and extract objects from the list. It allows the user to move about and manipulate the list in various ways, such as advance, backup, go to first, and go to a specific element in the list. List status operations are also provided.

MSG_LIST_PKG: An instantiation of the DOUBLE_LIST package for MSG_TYPE defined in MSG_DEF_PKG. Used to define linked lists of messages. The linked lists store messages for transmission until they can be placed in a specified users mailbox.

Processes and Methodologies

The SMB Software development process focused around an Evolutionary Prototyping approach, under a tailored DoD-STD 2167A [5]. As will be detailed in the metrics section of this paper, this approach allows better risk management while allowing flexibility in the design. Further this approach combined with Ada software reuse provided a number of very important advantages. One is that it allowed the very rapid development of a concrete executable model of the proposed system. This was important as the sponsor and end users could then evaluate the actual behavior of the system against their expectations. Another advantage is that it allowed the sponsor to fund the initial work as risk reduction and then only fully fund ideas that actually work. Sponsor and end user reaction have been very favorable. This favorable reaction has taken much of the stress that developers usually have when trying to establish a requirements baseline from scratch.

As stated above, Object Oriented Design was used as the design methodology. This methodology fits very nicely with the Evolutionary Prototyping approach. This is due to its iterative nature and support for subsystems definition [6]. With each prototype the next iteration on the objects within the system can occur giving the needed next increment of functionality. Subsystems are used to decrease the complexity of the system. Object Oriented Design provided the method for the decomposition of the reused Ada software into these software subsystems. It also provided the methodology for the design of the Tasking.

Environment

One of the goals stated above was to support for a reusable component library. An Integrated Programming Support Environment (IPSE) has been assembled which functions as the local software repository, development facility and configuration management system. The heart of this IPSE is the excellent, state-of-the-art Rational R-1000 Series 400 Ada Development Environment. The Rational allows very tight coupling of all project tools and products and is integrated as shown in Figure 6.

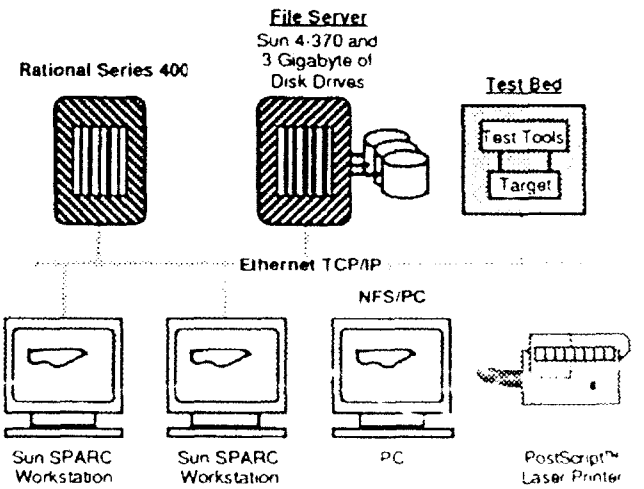


Figure 6. IPSE Integration

The Alsys Ada compiler for MS-DOS was used on the host PC with the Alsys Ada Cross-compiler for Intel processors used for the I/O boards. The compilers integrated with the Rational (through the use of Rational's Remote Compiler Integrator) have provided us with an unmatched capability for Ada software development.

Incremental Reviews

Using the Evolutionary Prototyping software development process often presented problems for meeting the requirements of Formal Reviews. Formal Reviews are defined by MIL-STD-1521 [7]. The purpose of these reviews are to provide the sponsoring agency with insight into the development necessary to determine that it is on track in terms of budget, schedule, and requirements. However MIL-STD 1521 specifies requirements for each review. As such no detailed design can occur before all the preliminary design is complete and the preliminary design cannot begin until all the system specifications are complete. This clearly is not suitable for the Evolutionary Prototyping software development process. Consequently we initiated the Incremental Design Review. The concept is that a Formal Design Review is satisfied incrementally by having several smaller User Evaluation Reviews. This is shown conceptually in Figure 7.

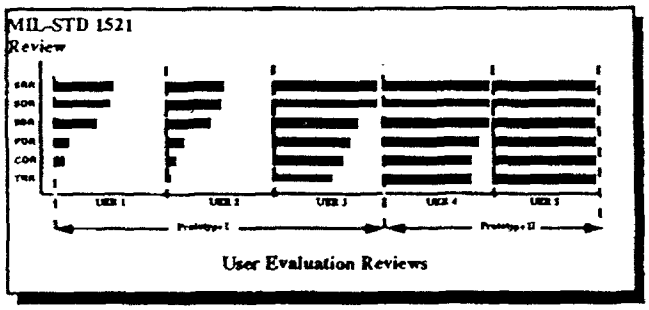


Figure 7. Incremental Reviews

Software Reuse Model

The software development reuse strategy for SMB was driven by minimizing software development costs through increasing productivity and quality [8]. However effective software reuse requires an architectural framework [9]. To put this framework into quantitative terms, a model of software reuse and its associated payoff as a function of software development phase was defined. This model is described as follows.

A software development consist of KDSI (Thousands of Delivered Source Instructions). Of that amount, let r be the maximum amount of reusable software defined on the interval (0, 1.0). Here software reuse includes requirements, documentation, design, code and testing. Software reuse is rarely accomplished, if ever, without some redesign, recoding or integration and testing; this effort is encapsulated in the Adaptation Adjustment Factor (AAF). (The notation in this section follows [11]).

Thus the AAF transforms the r*KDSI into an Equivalent DSI (EDSI) and the cost of developing the software for the Embedded Mode is:

$$3.2[(1-r)KDSI)^{1.2} + (r*EDSI)^{1.2}]$$

The AAF also encapsulates the behavior of software reuse as a function of life cycle. At the beginning of the software development, the factors effecting the AAF are most favorable. Factors which normally limit reuse are typically most lenient at the beginning of the life cycle. As the development progresses, introducing software reuse becomes more difficult. Consequently the amount of design modification, (DM) code modification (CM) and integration and testing (IT) increases as the development ages. As shown in Table 1, the AAF was very conservatively allowed to age from 13% to 33%. Introducing reuse at the end of the Test Readiness Review or Functional Qualification Testing is meaningless in our model. The results of our calculations are shown in Figure 8 for reuse ratios of 20, 50 and 80%.

End Review	DM	CM	IT	AAF
SSR	0.05	0.05	0.3	0.13
PDR	0.15	0.15	0.4	0.23
CDR	0.25	0.25	0.5	0.33
TRR	1	1	1	1
FQT	1	1	1	1

Table 1. AAFs versus End Reviews

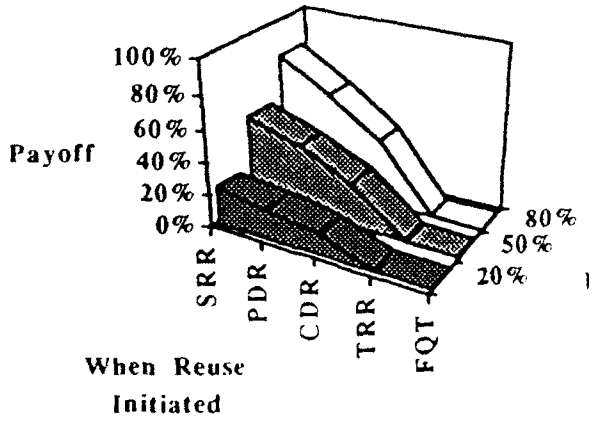


Figure 8. Reuse Payoff

Phase activity distributions were allocated as 6%, 16%, 25%, 40%, 19% for the phases ending in the System Specification Review (SSR), Preliminary Design Review (PDR), Critical Design Review (CDR), Test Readiness Review (TRR) and Formal Qualification Testing (FQT) respectively. The payoff is defined as a percentage of cost savings when compared with a normal development not using any software reuse. Thus payoff normalizes out several of the calibration factors. These calculations show

that the potential cost savings drops to less than half of the maximum amount possible if you wait until Critical Design Review to start searching for code to reuse. Alternatively, you can double your cost savings by implementing reuse before the end of System Specification Review.

Project Metrics

Size and effort estimation techniques have been used extensively during the software development process. The size metric is based on terminal semicolons, (TSC), and Delivered Source Instructions (DSI). For the original JAMES software (referred to in [3] as JAMH) which formed the core of the SMB, there were 9,379 TSC and 17,437 DSI.

In December 1988 the first size estimate of SMB based on Function Point Analysis [10] produced an estimate of 2,200 TSC (or 4,100 DSI) with a high and low of 2,870 and 1,550 TSC respectively and 2,060 TSC using the Delphi Expert Method [11]. The additional functionality required to reach Prototype II is listed in Table 2. Table 2 identifies the estimated size of each function in TSCs obtained from the Delphi Expert Method and its corresponding difficulty, D, to develop and integrate each function with respect to the existing software. The difficulty attribute was assumed to indicate risk. The difficulty attribute rating ranged from Easy, (E), Nominal (N), or Hard (H).

The sizing estimate obtained from Function Point Analysis does not lend itself to be partitioned into the same functionality groupings with which the users are familiar. For example, a Function Point is based on the number of input files, data fields and output files. Thus it becomes very difficult to map Function Points into the functions shown in Table 2.

Over the following six months, as Prototype I was tested and demonstrated, the functional requirements were re-evaluated during User Evaluation Meetings. Functionality was added, deleted, modified and re prioritized. This revision process was permitted so long as the total effort and risk remained nearly constant. In fact knowing the inherent uncertainties in this process, we required the effort and risk be smaller than the original estimate given in Table 2.

Users submitted the form shown in Figure 9 at the User Evaluation Reviews. Subsequently, functional requirements were analyzed and size and difficulty estimates were derived for each functional requirement and presented to the program sponsor. Using this data, the sponsor redirected Prototype II consistent with keeping the total effort and risk less than the original estimates. Table 3 shows the results of this process as baselined for Prototype II. The list of functions desired by the users produced a table many pages long. This table shows that the estimate of the total effort was reasonably accurate, within 20%, but sizing the individual functions had a high variance. This indicates the risk in trading off one function against another.

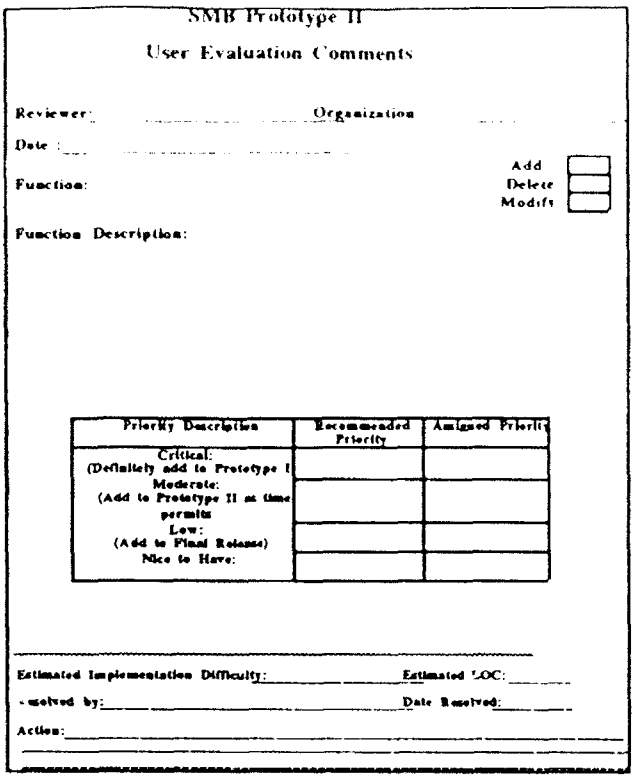
In the original cost analysis, the software development required to incorporate the JAMES software and provide the functionality shown in Table 2 was estimated to be $540K (1990 dollars) over a 22 month development schedule [12]. The cost of a new development without the benefit of the software reuse was estimated to be $2,600K. Based on this estimate, $545K was allocated over 24 months. After 17 months into the development schedule, Prototype II software development was completed and successfully tested. The remaining seven months will be used to bring the software from Prototype II to a formal product, which includes completing the DoD-STD 2167A documentation.

	Function	TSC	D
1.	Memory Management	200	H
2.	ASCII-Baudot Conversion	10	E
3.	I/O Port Setup	100	E
4.	Receive Message Buffer Flush	50	E
5.	Variable Length Message Lines	50	N
6.	Blank Message Form	50	H
7.	SPECAT Message Handling	50	N
8.	Visible Non-Printing Edit characters	100	N
9.	Message Catalog Sorting	750	N
10.	Message Format Identification	200	N
11.	Message Type Identification	50	E
12.	Message Parsing	100	N
13.	File Wipe	100	H
14.	Floppy Disk Access	50	E
15.	SA 2112 Switch Control	200	N
	Total	**2,060**	

Table 2. Initial Requirements and Estimated TSC for Prototype II

	Function	Est TSC	D	TSC
1.	Memory Management	200	H	514
2.	ASCII-Baudot Conversion	10	E	108
3.	I/O Port Setup	100	E	371
4.	Receive Message Buffer Flush	50	E	19
5.	Variable Length Message Lines	50	N	1
6.	Message Parsing	100	N	871
8.	Message Format Identification SSIXS Message Processing	200 100	N	
7.	Message Catalog Sorting	750	N	98
9.	Floppy Disk Access	50	E	68
10.	Security	50	N	68
11.	Duplicate Search	200	N	47
	Total	1,760		2,165

Table 3. Final Requirements and Estimated vs. Actual TSC for Prototype II



SMB Prototype II

User Evaluation Comments

Reviewer: Organization

Date:

Function: Add / Delete / Modify

Function Description:

Priority Description	Recommended Priority	Assigned Priority
Critical: (Definitely add to Prototype I		
Moderate: (Add to Prototype II as time permits		
Low: (Add to Final Release)		
Nice to Have:		

Estimated Implementation Difficulty: Estimated LOC:

Resolved by: Date Resolved:

Action:

Figure 9. User Evaluation Form

Lessons Learned

PC class computers are capable of the same functionality inherent in expensive minicomputers currently used in the DoD for message processing.

Ada can be used on PC class computers and still meet real-time message processing requirements.

Ada software reuse can significantly decrease program costs, but only if combined with a well defined software development process.

Partnership with Ada Joint Program Office can greatly enhance the Ada Software Engineering solution to difficult problems.

Future Directions

DoD requirements and the need for more flexible real time Command, Control and Communications systems are steadily increasing the need for information and computer security. This translates into software with Mandatory Protection (B level) certification and above [13]. Traditionally the more trust needed in one of these systems the higher the costs. The highest level systems certified for use at B level and above represent investments of millions of dollars and years of time for development. The need for a low cost, rapid development solution to this problem is required. The NRaD Operational Systems Branch is currently working on a solution to this problem. A set of bindings to the secure PC version of System V UNIX, Trusted XENIX, is being developed in partnership with the AJPO and Space and Naval Warfare Systems Command. In conjunction with the already developed IO TOOLS, this software will provide a real time B2 level secure system. Further this will lower risk to make reuse of Ada in future Trusted System successful.

Summary

This paper has discussed the processes and methodologies that have gone into the making of an embedded real-time system from software reuse. Evolutionary Prototyping combined with Object Oriented Design has proven very effective for the implementation of the Submarine Message Buffer System. Further this effectiveness has translated directly into shorter development times and substantial cost savings. However no process is a "magic bullet". The success achieved during the development of this system is in a large part due to the experience, training and dedication of the development team, both at NRaD and SPAWAR.

References

[1] Buchanan, G., Prisckin, J., Carlson, S., A Review of the Ninth Annual National Conference on Ada Technology, Ada Information Clearinghouse Newsletter, Vol. IX, No. 1, March 1991

[2] Keller, J., Special Report: 10 Ada Successes, Military & Aerospace Electronics, Vol. 2, No. 8, August 1991

[3] Lawler, J.M., Barlin, B. and Smith, R.A., Rapid Prototyping of Message Processor Systems, IEEE Military Communications Conference Proceedings, September 1990.

[4] Booch, G., Software Engineering with Ada, 2nd ed., The Benjamin/Cummings Publishing Co., Inc., 1987.

[5] Connell, J.L., Shafer, L.B., Structured Rapid Prototyping, Prentice-Hall, Inc., 1989

[6] Booch, G., Object Oriented Design with Applications, The Benjamin/Cummings Publishing Co., Inc., 1991

[7] Department of Defense, Technical Reviews and Audits for Systems, Equipments, and Computer Software, MIL-STD-1521

[8] Freeman, P., Reusable Software Engineering: Concepts and Research Directions, ITT Workshop on Reusability in Programming Proceedings, May 1983

[9] Kendall, R.C., An Architecture of Reusability in Programming, ITT Workshop on Reusability in Programming Proceedings, May 1983

[10] Albrecht, A.J. and Gaffney, J.E., Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation, IEEE Trans. Software Engineering, Vol. SE-9, No.6, November 1983

[11] Boehm, B.W., Software Engineering Economics, Prentice-Hall, Inc., 1981

[12] Air Force Contract Management Division, REVIC Software Cost Model, Version 8.7, Kirtland Air Force Base, New Mexico, August 1989

[13] Department of Defense, DoD Trusted Computer Systems Evaluation Criteria, DoD 5200.28-STD, December 1985

Author Biographies

Ben Barlin is the head of the Open Systems Development Branch at the Naval Command, Control and Ocean Surveillance Center, Research, Development, Test and Evaluation Division. He has worked on a number of Ada developments as both a software engineer and a software project manager. His research interests include software process improvement and Ada reuse. He has a B.S. in Computer Engineering from the University of California. He is a member of the IEEE Computer Society.

Jim Lawler is a Lead Engineer for the MITRE Corporation, Naval Warfare Systems Division. He has worked as a senior systems and software engineer on numerous government projects. His research interests include software process improvement and software metrics. He has a B.S. in Mathematics from the University of New York and an M.S. in Physics from the Virginia Polytechnic Institute. He is a member of the IEEE Computer Society.